LArSoft continuous integration system requirements

Mark Dykstra, Patrick Gartung, Lynn Garren, Marc Mengel,
Gianluca Petrillo, Erica Snider
Fermilab

Version 0.7
Aug 27, 2014

Abstract

This document will outline definitions and requirements for a "Continuous Integration" build and test system for LArSoft and related experiments, which will automatically build or test the LArSoft or experiment software when new updates are pushed into the central version control system (VCS) for related software packages. The system uses the Central Build Service being provided by the Fermilab Scientific Computing Division.

1. Assumptions

The requirements and specifications listed below are built on the assumption that the available continuous integration (CI) system will conform to the Central Build Service system requirements as documented in CD-DocDB-5319.

2. Definitions

These are terms used later in the document:

CI system	The set of tools, applications and machines that provide for specification and automated execution of tests of LArSoft software
CI application:	The software application that automates CI workflow triggering and execution
Build step:	the smallest increment of execution provided by the CI application
Project:	A set of build steps that the CI application runs as a single, self-contained unit.
CI job:	The execution phase of a project
Trigger:	An event that initiates execution of a project. Triggers specify the values of run-time configuration parameters.

Integration task:	The basic unit of CI testing available to end-users via triggers
Workflow:	A set of steps executed as part of an integration task. The steps may be part of a single project, or a set of projects that are run in a sequence specified by triggers defined internally to the set of projects in the workflow.

3. CI System Requirements

3.1. Triggering

3.1.1. Trigger input specification

It must be possible to specify parameters in the trigger that can be used by the triggered CI jobs to determine run-time configuration, such as input data and configuration data sources used in a project.

a) Note that projects may also specify input and configuration data sources within the project definition that are independent of any trigger parameters

b) All relevant configuration information in a trigger event must be propagated to subsequent triggers within a given workflow.

3.1.2. Dynamic parameters

(*Desirable*) A project should be able to define parameter values at run time that can then be used in triggers called within the project.

3.1.3. Input parameter validation

The system should validate input parameters prior to attempting to queue or run a CI job.

3.1.4. Delay

There should be a configurable delay between the time of a trigger event and the launch of the triggered workflow.

3.1.5. Actions

There must be able to be triggers on at least the following actions:

a. push to develop branch on central repository, which will initiate a build and test of code on the head of the develop branch
b. push to master branch on central repository, which will initiate a build and test of code on the head of the master branch
c. successful completion of nightly build
d. successful completion of integration release build (if automated with the CI system)

e. successful completion of production release build (if automated with the CI system)
f. arbitrary manual triggers

3.1.6. Manual triggering mechanism

It must be possible for authorized users to manually trigger projects or workflows via command line or GUI

3.2. Logging

3.2.1. Archive input/config

It must be possible to archive input and configuration data in a manner that supports the same level of version control as that used by the software being tested.

a) There must be a tool or suitably easy method for understanding the mapping of this versioned data to the specific versions of code to which is applies, and visa versa.

3.2.2. Tags

The system should provide a unique, easily distinguishable tag that identifies all CI jobs that were run and results obtained within a given workflow.

a) The tagging scheme should also identify the test suite, as appropriate.

3.3. Data Transfers

3.3.1. Inputs

Input data may be in the form of parameter values passed to the project with the trigger, or data files passed via some generally available (on all relevant platforms), grid-compliant transfer protocol.

a) There should be a mechanism for local admins of build slave nodes to specify the technology to use for these data transfers on a per-slave basis.
b) A CI job must have access to the output from previous build steps and CI jobs in a given workflow, or some previously run workflow.
c) The test configuration may specify input data files. All specified input files will be fetched by the CI system prior to test execution. Users must be able to override any of these file specifications on the triggering command line or GUI

3.3.2. Outputs

Output files must be transferred via generally available (on all relevant platforms) grid compliant transfer protocols. If a given project triggers another project, output data may be exported to the triggered project via trigger parameter.

a) Log files and summary files from tests should be transferred to a location where they can be accessible via the CI application web interface.

b) Output data files can be declared in the test configuration for the purpose of local sanity checks. The CI system will leave these files in place and will not take responsibility for transferring them to another location.

3.3.3. Credentials

The system must manage any credentials required to perform input and output data transfers.

3.3.4. Encryption

All data channels, including parameters passed via triggers, must be encrypted.

3.3.5. Access to output data within tests

Any given test should be able to access output data from previous tests or CI jobs in a given workflow, or from previously run workflows.

3.4. Workflows

3.4.1. Support for software versions

Workflows that build software must support custom, per-build specification the branch, tag or commit to use for each repository to be included in the build. For a workflow that involves multiple repositories, the system must:

a. Allow specification of a single "global" branch or tag, which directs the system to use that branch or tag in every repository in which it is found.
b. Use the following scheme to resolve branches, tags or commits to use in a build:
    i. If a branch, tag or commit has been specified for a given repository, build the specified code in that repository. Failure to find the specified branch, tag or commit should result in an error condition.
    ii. If no branch, tag or commit has been specified for a given repository, use the global branch or tag if defined and present in the repository.
    iii. If no global branch or tag has not been specified or is not found in the given repository, then default to the head of the develop branch.
c. Pull software to be built from some combination the following: the central repository, from some other secure repository, or from a secure distribution source of tar files that are then unwound in situ.

(*Desirable*) For LArSoft builds, it would be useful to have a command that examines the "active" branches in a user's work area, then launches a build of those branches on the central repository.

3.5. Independence

3.5.1. Tests

All tests must be executable independently of the CI service.

3.5.2. Workflows

All workflows must be executable independently of the CI service.

3.5.3. Builds

All Builds must be executable independently of the CI service.

3.5.4. Site independence

Aside from CI application functions, which are assumed to reside with the Central Build Service operated by Fermilab, all functionality of the system should be available on any supported platform running at any network-accessible site.

3.6. Integration/Development

3.6.1. Migration

The system should allow for simple migration of workflows and projects from a development/integration environment into production

a) No changes to the test scripts or CI application configuration should be needed in order to accomplish this migration.
b) Results from projects or workflows under development or integration testing should be visible from the production CI application.

3.7. Platform support

3.7.1. All Supported

All CI jobs and workflows must run on all supported platforms at any site that runs any of those platforms.

3.8. Code management

3.8.1. Infrastructure software repository

All CI system infrastructure software will live in a repository / VCS.

3.8.2. Infrastructure software versioning

The versioning used for CI system infrastructure software will be no less finely grained than that of the software that must be tested.

3.8.3. Distribution

The CI system infrastructure software must be configured to allow distribution via a ups product (although distribution via ups is not a requirement).

3.9. Test execution support

3.9.1. Dynamic test identification

The CI system must have a dynamic mechanism for identifying the proper tests to execute at run time.

a) It must be possible to add or remove tests without modifying the CI application configuration or any CI system scripts.

3.9.2. Stand-alone test format

Stand-alone tests must be in the form of an executable script located under a dedicated test directory in the appropriate software repository / VCS.

3.9.3. Unit test format

Unit tests will make use of the “cet_test” macro provided by the cetbuildtools package. The build system will perform the required dynamic identification and execution sequencing for these tests.

3.9.4. Test log files

Standard output and standard error from each stand-alone test script should be logged to a file managed by the CI system. These files should be

3.9.5. Suites

The CI system must support a mechanism for associating specified groups of tests that define a “test suite”. A single trigger event should result in the execution of a single test suite.

a) A test suite may trigger other test suites as part of a workflow.

3.9.6. Per-test time estimates

(Desirable) Tests should have an optional standardized execution time score that allows the system to estimate the CPU required to run the test on a given platform.

3.9.7. Timeouts

The system should provide a means to force test failure if the run time is:

a. below some user-defined, per-test minimum CPU time;
b. above some user-defined, per-test maximum CPU time threshold that is scaled to the time score;
c. above some user-defined, per-test maximum absolute CPU time threshold;
d. or above some system-defined, per platform absolute CPU time threshold

3.9.8. Per-test memory estimates

Tests should have an optional memory score allows the system to estimate the expected maximum memory size of the test during execution.

3.9.9. Memory limits

The system should provide a means to force a test to fail if the memory consumption is:

e. above some user-defined, per-test maximum memory size threshold that is scaled to the memory score
f. above some user-defined, per-test absolute memory size threshold
g. above some system-defined, per platform absolute memory size threshold

3.9.10. Data management

Input and output data, and all log files created on the test execution machine should be stored in a way that allows easy identification of the test to which it pertains, and the test execution instance that used or generated the data.

3.9.11. Parallelism

The CI system should allow parallel execution of tests when possible

3.10. Basic testing support functions

The following additional functionality will be integrated into the CI infrastructure that executes stand-alone tests

3.10.1. Output file sanity checks

For declared output files, the CI system should provide the following checks:

a) Existence of the file
b) Checks against a specified minimum and maximum file size
c) (Desired) Readability of the file by root (if with a ".root" extension)

Files that fail any of these checks will cause the test to be marked as "failed".

3.10.2. Standard art logfile scanning

All test logfiles that are determined to be art output logs will be scanned to ensure that a set of essential lines are present, and that no lines suggesting severe error conditions occurred. Test logs the fail the scale will cause the test to be marked as “failed”.

3.10.3. Histogram comparison checks

(Desirable) A utility should be provided that can be used within a test to compare a set of histograms in two files. The comparison should allow the test to fail if the comparison fails some minimum criterion of agreement.

3.10.4. Data logging

All test logfiles will be scanned for CPU and memory consumption data.

When enabled by the appropriate trigger parameters, the following data from each test executed in a workflow will be logged to a database:

a) The CPU and wall clock time required to run the test
b) The CPU time spent in each module
c) The peak memory consumption for the program
d) (Desired) The memory consumption as a function of time during execution of the program.
e) The test name
f) The test suite name
g) The workflow instance tag
h) Date and time that the workflow was run
i) Software product and version being tested, including any branch, tag or commit information as specified in the workflow trigger.
j) CI software version being used
k) The machine on which the machine was run
l) The CPU scale factor for this machine
m) The names and sizes of all output files

Failures in data logging should not cause a test to return a “failed” status.

3.11. Display

3.11.1. Real-time execution monitoring

(Desirable) Real-time visualization of build-step progress and results within a project and results within a workflow is highly desirable.

3.11.2. Recorded data visualization

(Desirable) Plots of the following data should be provided via some means:

a) The historical CPU times, memory consumption and file sizes for a given test in a given test suite

b) The memory consumption as a function of time for a given test instance

3.11.3. Test result summaries

A summary of test results should be available via the CI application web interface. This summary should provide a summary of the following information:

a) Pass / fail for the entire workflow

b) Pass / fail for each individual CI job within the workflow

c) Pass / fail for each individual test within a CI job

d) For failed tests, which phase of testing produced the error (e.g., input file retrieval error, error status returned from test script, exceeded CPU time limit, logfile scan error, etc.)

3.11.4. Test log files

Log files from test scripts and workflows must be easily accessible via the web.